REPORT DOCUMENTATION PAGE

AD-A213 786

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
IBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): ARO 23785.3-EL-F
6a. NAME OF PERFORMING ORGANIZATION: Univ of North Carolina	6b. OFFICE SYMBOL (If applicable):
	7a. NAME OF MONITORING ORGANIZATION: U. S. Army Research Office
6c. ADDRESS (City, State, and ZIP Code): Chapel Hill, NC 27514	7b. ADDRESS (City, State, and ZIP Code): P. O. Box 12211 Research Triangle Park, NC 27709-2211
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: U. S. Army Research Office	8b. OFFICE SYMBOL (If applicable):
	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: DAAL03-86-G-0050
8c. ADDRESS (City, State, and ZIP Code): P. O. Box 12211 Research Triangle Park, NC 27709-2211	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification): A Test Bed for a Massively Parallel Computer Architecture

12 PERSONAL AUTHOR(S): Gyula Mago

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 8/1/86 TO 7/31/89	October 1989	4

16. SUPPLEMENTARY NOTATION: The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Parallel Computer Architecture, Graph Reduction, Programming Languages, Lambda Calculus

19. ABSTRACT

Mr. Partain has carried out research on alternatives to graph reduction as it is commonly understood. His main objective -- which he has reached -- was to develop a way to do graph reduction without actually using pointers. This will permit parallel implementation of functional programming languages without having to rely on a global address space. Given some suitable machine primitives, such an implementation is shown by Partain to be competitive with ordinary graph reduction. From a mathematical point of view, the results provide a way to do lazy normal-order tree reduction for the lambda calculus.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

ARO 23785.3-EL-F

FINAL REPORT

1. ARO proposal number: 23785-EL-F

2. Period covered by report: 1 August 1986 -- 31 July 1989

3. Title of proposal: *A test bed for a massively parallel computer architecture*

4. Contract or grant number: DAAL03-86-G-0050

5. Name of institution: University of North Carolina at Chapel Hill

6. Author of report: Gyula Mago
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27514

7. Scientific personnel supported by this project :

This Army Science and Technology Fellowship was awarded to Mr. William Partain (SS# 456-31-0758), a graduate student in the Department of Computer Science at UNC in Chapel Hill.

SUMMARY OF RESEARCH FINDINGS

Mr. Partain has carried out research on alternatives to graph reduction as it is commonly understood. His main objective -- which he has reached -- was to develop a way to do graph reduction without actually using pointers. This will permit parallel implementation of functional programming languages without having to rely on a global address space. Given some suitable machine primitives, such an implementation is shown by Partain to be competitive with ordinary graph reduction. From a mathematical point of view, the results provide a way to do lazy normal-order tree reduction for the lambda calculus. (For more details, please see the enclosed pages.)

The final draft of the dissertation has been written, is currently being read by members of the dissertation committee, and he is expected to finish and defend his dissertation early in the Fall term of this academic year. He will earn the PhD degree from UNC, Chapel Hill in Computer Science. Publications reporting on this work will be produced subsequent to the defense.

Graph Reduction Without Pointers

William D. Partain*
(Under the direction of Prof. Gyula A. Magó)

Ph.D. dissertation to be submitted to
The University of North Carolina at Chapel Hill

September 28, 1989

Abstract

Graph reduction is one way to overcome the exponential space blow-ups that simple normal-order evaluation of the lambda-calculus is likely to suffer. The lambda-calculus underlies lazy functional programming languages, which offer hope for improved programmer productivity based on stronger mathematical underpinnings. Because functional languages seem well-suited to highly-parallel machine implementations, graph reduction is often chosen as the basis for these machines' designs.

Inherent to graph reduction is a commonly-accessible store holding nodes referenced through "pointers," unique global identifiers; graph operations cannot guarantee that nodes directly connected in the graph will be in nearby store locations. This absence of locality is inimical to parallel computers, which would prefer having isolated pieces of hardware working on self-contained parts of a program.

In my dissertation, I develop an alternate reduction system using "suspensions" (delayed substitutions), with terms represented as trees and variables by their binding indices (de Bruijn numbers). Global pointers do not exist and all operations, except searching for redexes, are entirely local. The system is provably equivalent to graph reduction, step for step. I show that, if this kind of interpreter is implemented on a highly-parallel machine that supports a locality-preserving linear program-representation and fast scan primitives (an FFP Machine is an appropriate architecture), then the interpreter's worst-case space complexity is the same as a graph reducer (that is, equivalent sharing) and its time complexity falls short on only one unimportant case. On the other side of the ledger, graph operations

*Supported by an Army Science and Technology Fellowship (grant number DAAL03-86-G-0050) administered by the U. S. Army Research Office.

that involve chaining through many pointers are often replaced with a single associative-matching operation. What is more, this system has no difficulty with free variables in redexes and is good for reduction to full beta-normal form.

These results suggest that non-naive tree reduction is an approach to supporting functional programming that a parallel-computer architect should not overlook.

1 Basic results

The basic results in my dissertation are:

- A suspension-based λ_s-interpreter is a correct implementation of the pure λ-calculus because its manipulations of λ_s-terms are isomorphic to the manipulations of λ_g-graphs by a graph-reducing λ_g-interpreter.
- When the λ_s-interpreter is implemented on an FFP Machine (FFPM) or similar architecture, its worst-case space complexity is within a constant factor of that of a lazy-copying graph-reducer on a global-addressable-memory (GAM) machine.
- The worst-case time complexity of the FFPM interpreter is equal to or better than that of the GAM interpreter, except for the last-instance relocations of suspension pointees.

I conclude that graph reduction does *not* have an inherent advantage as a computational model to support lazy functional programming. I now review the major issues raised by comparing the two interpreters.

2 Reduction to β-normal form

Graph reduction that uses lazy copying is only suited to reduction to weak β-normal form (WBNF); it cannot cope with free variables in redexes. Also, binding indices cannot be used with this kind of graph reduction. A graph reducer must either support α-conversion or use backpointers to avoid name-capture problems. Also, to enjoy maximal sharing with graph reduction, one must include expensive detection of maximal free expressions (MFEs) in each reduction step ("fully lazy" copying); when reducing to β-normal form (BNF), the less onerous "lazy" copying does *not* work.

Suspension-based reduction to BNF works with binding indices, and lazy copying is a completely natural mechanism. If an implementer wanted to use a reduction order or normal form that allows free variables in redexes—e.g.,

innermost spine reduction or BNF—then suspension-based reduction steps to the front of the class.

Oddly, the FFPM implementation that does so well with suspension-based reduction to BNF finds reduction to WBNF *more* costly. Algorithms for finding out if a term is inside a λ-abstraction are ill-matched to what the hardware can do well.

3 Linear representation

The attractions of suspension-based reduction to computer architects center around a linear representation of program symbols. A symbol need *not* be globally addressable nor stored in a global resource.

Binding indices, used to avoid name-capture problems, are very amenable to manipulation with fast scan primitives. For example, all bound variables in a λ_s-term can be detected in $O(\lg n)$ time.

The use of associative matching to detect redexes and other "interesting" patterns of symbols is noteworthy. This matching can find a redex anywhere in a λ-term in as little as one step, whereas a graph reducer must necessarily chain through pointers to get there. An important aspect of this matching (and the other FFPM algorithms) is the modest amount of "parse-tree" information that must be synthesized from the raw symbols—no more than two selectors are ever needed.

I think it worthwhile to have presented a sizable example using the low-level techniques possible on an FFPM. I believe that these techniques would work just as well on *any* parallel architecture with fast scans and a locality-preserving linear program-representation.

The cost of a linear program-representation (besides the implementation cost) is "last-instance relocation," which means that the last copy of a λ_s-abstraction must be moved into place. The analysis of this movement shows how constrained the "extra" copying of tree reduction can be.

4 The next step

I hope I have laid to rest the notion that "string" reduction is inherently, wildly inefficient for normal-order reduction. The basic question that follows is: Can the λ_s-interpreter be "grown" into a practical mechanism for the efficient execution of lazy functional programs? Intimately related to this question are the questions of what realistic functional programs actually do ("what happens above") and the constraints and properties of the

target hardware architecture ("what happens below"). The truly successful architect for a parallel reduction machine will be master of all of these levels.

I sketch several ways in which a λ_s-interpreter could be extended for practical use. Suspension lists seem a clear winner, and Révész's extensions to integrate lists directly into the calculus are no less intriguing. I would allow bound-variables in suspension pointees and use that to implement recursion. I would do some speculative copying based on simple heuristics derived from real programs: an example might be, "if filling a suspension's next-but-last bound variable, fill the last one as well."